ERDC MSRC PET Preprint No. 01-32

Minimizing Runtime Performance Variation with Cpusets on the SGI Origin 3800

by

Jeff Hensley, Robert Alter, Daniel Duffy, Mark Fahey, Lee Higbie
Tom Oppe, William Ward, Marty Bullock, Jeff Becklehimer

9 October 2001

Work funded by the Department of Defense
High Performance Computing Modernization Program
U.S. Army Engineer Research and Development Center
Major Shared Resource Center through

Programming Environment and Training

Supported by Contract Number: DAHC94-96-C0002
Computer Sciences Corporation

Minimizing Runtime Performance Variation with Cpusets on the SGI Origin 3800

Jeff Hensley[*], Robert Alter[†], Daniel Duffy[*], Mark Fahey[*], Lee Higbie[*],
Tom Oppe[‡], and William Ward[*]
Computational Science and Engineering Group
U.S. Army Engineer Research and Development Center
Major Shared Resource Center
Vicksburg, MS

Marty Bullock and Jeff Becklehimer
SGI Federal
U.S. Army Engineer Research and Development Center
Major Shared Resource Center
Vicksburg, MS

Abstract

The U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC) currently employs one of the largest SGI Origin 3800 (O3K) systems in production use. With more than 100 users, the system has provided an extremely robust and stable environment with good throughput. The system operates under a high load average and users have experienced large differences in the run times of their jobs.

In order to better understand these disparities, a series of benchmark tests comprised of application codes was conducted on a 256-processor O3K to study the system performance both with and without the use of cpusets. The results show a large variation in timings of individual jobs that were run without cpusets on the loaded system. With the implementation of cpusets, this large variation was eliminated and the overall system performance was greatly increased.

1 Introduction

One of the largest SGI Origin 3800 (O3K) systems in production use is located at the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC). This O3K has 512 processors configured in a Single System Image (SSI). An O3K is composed of compute bricks (C-bricks) interconnected by router bricks (R-bricks). In the machines at the ERDC, each C-brick has four 400 MHz R12K processors and four Gbytes of memory. User jobs are queued by submitting them to the Portable Batch System (PBS) batch queuing system.

[†] Alpha Data Corporation
[*] Computer Sciences Corporation
[‡] Data Management Consultants

Since the system was brought into production status as an SSI, there have been reports from users about inconsistencies in the time required to perform jobs. Some users reported that the same job could take as much as five times longer to complete during one run as during another.

Diagnosing the cause of the problem is difficult because of the inconsistencies of reported results; one user might perceive a slowdown in performance while another user might not. Furthermore, depending on the state/load of the machine, the same job run at different times could result in significantly different run times. Several possible explanations for these results are as follows:

1. *Competition with system resources.* As the O3K is currently configured, user, system, and I/O processes must compete for the same resources. On a system fully loaded with user jobs, it is quite likely that this leads to competition between system resources and user tasks, resulting in the time-sharing of processors for various tasks.

2. *Competition for memory.* From the user's vantage, the O3K has a single address space. Even though each C-brick has 4 Gbytes of memory (1 Gbyte/processor), processes can request and obtain much more memory than 1 Gbyte per processor. Hence, even though multiple processes may be running on the same C-brick, several of these processes may have to access memory located on another C-brick.

3. *Migrating processes.* During execution of a job, the processes are not tightly bound to a processor and its local memory. The operating system can move these processes around, resulting in nonlocal (off-node) memory requests.

One way to address these issues is through the use of cpusets. A cpuset is a designated collection of processors that form a separate envelope of computational resources (processors and/or memory). System resources can effectively be "soft" partitioned to prevent runtime competition for resources by various tasks.

SGI provided the authors of this report access to a 256-processor O3K at the ERDC. This machine was originally configured identically to the 512-processor machine, i.e., without cpusets. The Computational Science and Engineering (CS&E) group at the ERDC MSRC developed a series of benchmark tests to run on this system. The benchmark tests were run on the system under the original configuration (without cpusets) and then again after the system was reconfigured with cpusets.

Section 2 of this paper describes cpusets and the implementation used for this study. Section 3 describes the throughput tests and discusses the results of these tests. Section 4 deals with an additional test that was run to study the effect of jobs requesting very large numbers of processors. A summary and conclusions are provided in Section 5.

2 Description of Cpusets

On the O3K, cpusets can be configured to partition system resources and restrict access to these resources. Cpusets are dynamic in the sense that they can be created and deleted without a system reboot. A commonly accepted method of configuring cpusets is to create a “boot cpuset” at system startup that remains static during normal operation. This boot cpuset is used to run the operating system and associated daemons. The remaining processors are then given to the batch scheduler to run user jobs. The batch system can then dynamically create a cpuset for each user job at startup and delete the cpuset upon job completion.

At creation, a cpuset is designated as either exclusive or nonexclusive. Processors that are members of an exclusive cpuset can only be utilized through the use of the *cpuset* command. Nonexclusive cpuset processors can be used through both the cpuset interface and through normal system scheduling.

For this study, the system was reconfigured to have a boot cpuset and to use dynamic cpusets for user job execution. The boot cpuset, which isolates the kernel processes and daemons from the users’ jobs, consisted of two C-bricks (8 processors) for this study. This cpuset is crucial to eliminating one source of resource contention. See Figure 1 for a schematic representation of the cpuset configuration on the system.

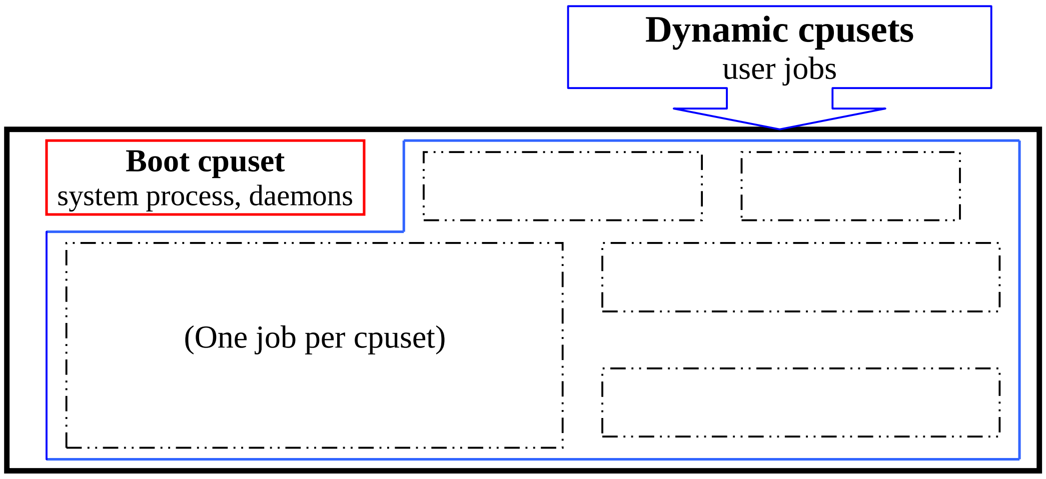

Figure 1. Schematic representation of cpusets on the O3K.

It is important to note that, as implemented for this study, PBS defines cpusets in terms of C-bricks, so that each cpuset has a multiple of four processors and a given job has exclusive access to the resources of the cpuset. This means that each user job must request system resources in terms of C-bricks, not processors. A user wanting to run a serial job would need to request the entire C-brick (4 processors). For optimal system usage, the user must be aware of this issue.

Also note that the cpusets were configured to include the memory of each C-brick, so that the memory available to a given job is the total memory in the cpuset (that is, 4 Gbytes times the number of C-bricks).

3 Benchmark Tests

A throughput benchmark test was developed that consisted of jobs using six different user application codes; all of the codes were MPI codes. Different input sets and different processor counts were used to create 15 different jobs, and some of these jobs were run multiple times in a given throughput test. This resulted in a total of 39 tasks in the throughput test.

The throughput test was run three times while the system was configured without cpusets and three times while the system was configured with cpusets. Additionally, each problem was run, without cpusets, as a dedicated test multiple times (twice in some cases, three times in others) to provide a reference for optimum time. The jobs were submitted to the batch queue (using PBS) in each of the throughput tests and the order of job submission was the same in each test.

The timings for the various runs are given in Table 1. A baseline dedicated time is given for each job. The reference time for the dedicated test is an average over multiple (at least two and usually three) dedicated runs. In fact, this average was taken for dedicated runs while the system was configured without cpusets. The times were, in general, marginally less for dedicated runs when the system was configured with cpusets.[1] The last row of the table gives the total wallclock time of all of the jobs comprising each throughput test, while the entry in the dedicated column provides the total time if all of the jobs had completed in the amount of time required for the individual dedicated runs.

Looking at the total times, one can easily see the large variation in performance for the three tests without cpusets. The observed variation between test 1 (87,591 s) and test 3 (117,473 s) is a relative difference of approximately 34 percent.

The observed variation of time for identical jobs when running without cpusets is quite striking. Consider, for example, prob_m. This 32-processor job was run six times in each throughput test, and, in the absence of cpusets, there is a remarkable variation in the observed execution times, both within a given throughput test and across throughput tests. The minimum time recorded for this task (without cpusets) was 2,514 s (which is essentially the dedicated time), but one instance of prob_m actually took 13,321 s, or more than five times as long. Figure 2 graphically illustrates the timing variation for prob_m, while Figure 3 shows the variation that occurred for prob_f.

[1] One possible explanation for this minor variation is that without cpusets there may still be some modest conflict between the user job and the system tasks. This conflict might be eliminated when using cpusets. However, in tests of one code that was not part of the throughput tests, the time for dedicated runs was measurably faster (by approximately 10 percent) when *not* using cpusets. This code used MLP rather than MPI as all other codes did. The authors do not have any explanation for this result at this time.

In contrast to the timing fluctuations observed when cpusets were not in place is the consistency when the machine was configured to use cpusets. Timing variations are quite small and the timings corresponding to identical runs are normally within a few seconds of each other.

Finally, the total time required by the entire throughput test is markedly less when cpusets are used. Note that the ratio of the worst total time without cpusets (117,473 s) to the total time with cpusets (62,654 s) is approximately 1.9, meaning that the throughput time was cut nearly in half by using cpusets (while using fewer processors).

4 Additional Tests

Anecdotal evidence gathered by CS&E suggests that jobs run at large processor counts (i.e., jobs that request all or almost all of the processors) on the O3K at the ERDC MSRC may suffer some performance degradation. Two codes were selected to run jobs at large processor counts on the system while it was configured with cpusets and again while it was configured without cpusets. The largest number of processors available was 248 when the system was configured with cpusets; when cpusets were not used, it was possible to use all 256 processors.

Since it had been shown that individual runs with cpusets resulted in consistent amounts of wall-clock time used, the two codes were run only once with cpusets configured on 248 processors (this took into account the 8-processor boot cpuset). Additionally, these codes were run without a boot cpuset configured at both 248 and 256 processors counts. The results of these tests are shown in Table 2.

Remarkably, for these two codes, the results show that there is no advantage to using all 256 processors on the machine. In fact, it is to the user’s advantage to run within a cpuset with fewer processors while eliminating any contention with the system processes.

5 Conclusions

The results of these tests are convincing; if users’ jobs are scheduled on all the processors of an O3K there will be large variations in performance of individual jobs unless cpusets are configured. The magnitude of these variations can be quite striking, with some jobs taking as much as five times longer to run in some instances than in another. Although the data gathered here does not necessarily indicate what the problem is, it does appear that contention for system resources is the likely culprit.

These results strongly suggest that due consideration should be given to reconfiguring the 512-processor O3K at the ERDC MSRC to run with cpusets. The advantages and disadvantages of reconfiguring the O3K with cpusets are summarized as follows:

Advantages

1. The use of cpusets partitions jobs from each other. One user’s job will not compete with another user’s job (or system jobs) for the resources of a given C-brick.
2. The use of cpusets will ensure consistent performance.
3. The use of cpusets will enhance overall system performance.

Disadvantages

1. PBS requires that the user request processors in blocks of four. For example, a serial job running on one processor would actually require all four processors of one C-brick.
2. The user must understand that jobs requiring large amounts of memory will require the allocation of additional processors. If a job requires more than 1 Gbyte of memory per processor, then additional processors must be requested to obtain the necessary memory. For example, if a job needs 16 Gbytes of memory, it would require 16 processors even if it were an MPI job with four processes.
3. The configuration would use a boot cpuset consisting of (probably) eight processors. These processors would not be included in the compute pool.
4. Users would have to be educated about the above issues, particularly items #1 and #2 above.

The disadvantages certainly must be considered. However, the observed improvement in the throughput test with cpusets in place is striking; the worst throughput test without cpusets took nearly twice the time of the throughput tests with cpusets. Also, the inconsistency that exists on the loaded machine without cpusets can create confusion and concern for the users.

The most problematic concern is that jobs would be required to request processors in blocks of four. A user who wants to run a job on fewer than four processors may complain about being charged for more than his/her job is actually using. However, there are other possibilities, including different batch queuing systems, which could be explored.

The creation of a boot cpuset reduces the total number of processors available to run a particular job. For this study, the number of processors in the compute pool was reduced from 256 to 248. However, the test that was performed with large processor count jobs gives an indication that one might experience poorer performance when trying to use all of the processors on the machine (that is, without cpusets) than would be obtained by using fewer processors. In the event that there is a special need to use all of the resources of the machine, this could still be accomplished by using dedicated time on the machine and reconfiguring without cpusets after a reboot.

6 Future Tests

Resource allocation and job scheduling on a high-performance computer is an intricate task. While the results in this study strongly indicate that the use of cpusets can greatly enhance system performance, there are other issues to be considered.

One issue that merits further exploration is the effect of the parallel programming paradigm that is used. As mentioned above, all of the codes used in the throughput tests were MPI codes, but a code that used MLP was run in dedicated mode when the system was configured with cpusets and without cpusets. The times required with cpusets were approximately 10 percent longer than without cpusets. Future tests should include codes that use other parallel mechanisms, such as SHMEM and OpenMP.

	no. of		without cpusets			dedicated	with cpusets		
problem	cpus	copy	*test 1*	*test 2*	*test 3*		*test 1*	*test 2*	*test 3*
prob_a	128	1	3528	3707	2796	2802	2749	2753	2754
prob_b	8	1	2602	2727	2630	2626	2625	2625	2624
		2	2806	2907	3064		2623	2622	2624
		3	2646	2604	3967		2623	2624	2624
		4	2608	2605	2602		2623	2624	2624
		5	3496	4067	2601		2623	2624	2623
		6	2606	3681	2774		2628	2624	2624
		7	2880	5354	2603		2625	2624	2624
prob_c	32	1	726	728	2167	733	732	733	732
		2	729	730	1273		731	733	731
		3	730	729	732		732	731	732
		4	728	730	1400		733	732	731
		5	769	731	786		736	735	731
prob_d	128	1	6214	6296	4498	3234	3217	3214	3213
prob_e	32	1	4169	11040	3108	3046	3056	3053	3047
		2	3056	3065	13882		3055	3048	3046
prob_f	64	1	1384	5311	1616	1381	1383	1382	1382
		2	2763	2070	2789		1381	1382	1384
prob_g	64	1	772	428	619	393	393	394	417
		2	702	428	554		393	393	397
prob_h	16	1	766	774	765	765	763	763	765
		2	772	765	764		764	765	764
		3	779	769	781		765	766	765
prog_i	32	1	429	427	419	419	420	422	420
		2	424	422	423		422	422	420
		3	423	437	421		421	422	422
prob_j	64	1	254	255	251	249	248	249	250
		2	252	256	345		251	248	248
		3	256	258	260		250	249	250
prob_k	64	1	3186	2385	2728	2290	2287	2289	2287
prob_l	128	1	1280	1264	1267	1253	1251	1252	1251
prob_m	32	1	3408	2527	2514	2523	2516	2507	2500
		2	7671	10701	9496		2519	2504	2507
		3	3754	2623	13321		2521	2513	2504
		4	4238	9513	12442		2513	2507	2504
		5	7764	2528	9397		2511	2502	2507
		6	2629	2528	2522		2516	2513	2508
prob_n	112	1	2606	1420	2086	1352	1334	1332	1330
prob_o	224	1	786	794	810	832	786	785	788
TOTAL			**87591**	**100584**	**117473**	**62886**	**62719**	**62660**	**62654**

Table 1. Results of throughput tests. Times (in seconds) are recorded for the various problems (and multiple runs).

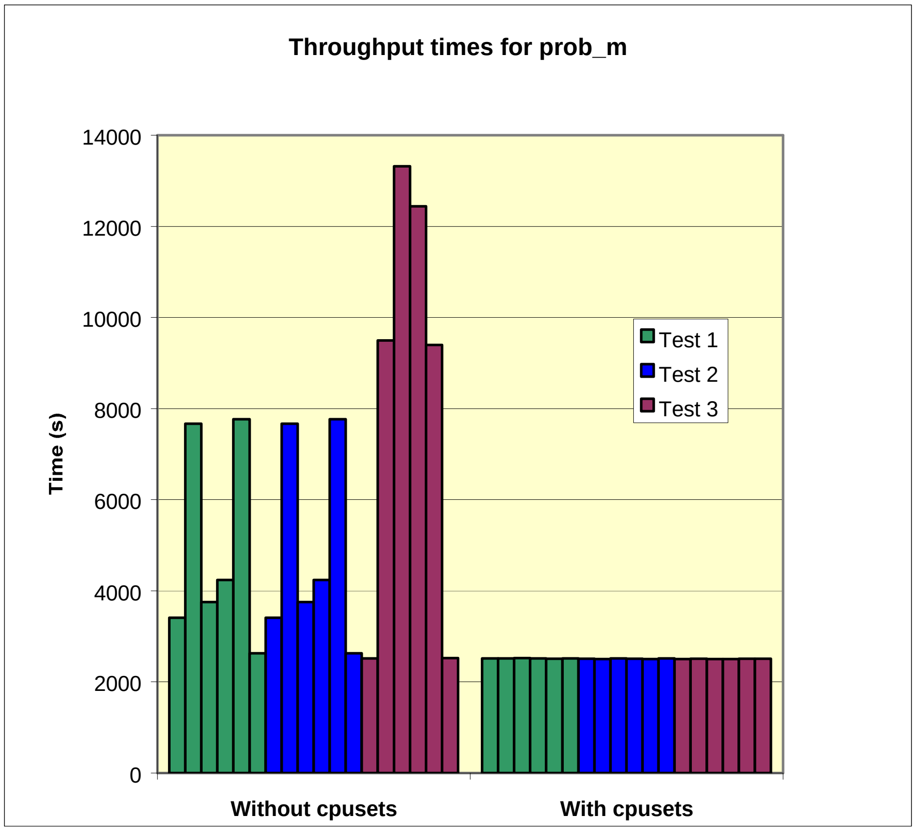

Figure 2. Throughput times for prob_m showing execution times for the six identical jobs that were part of each of the three throughput tests.

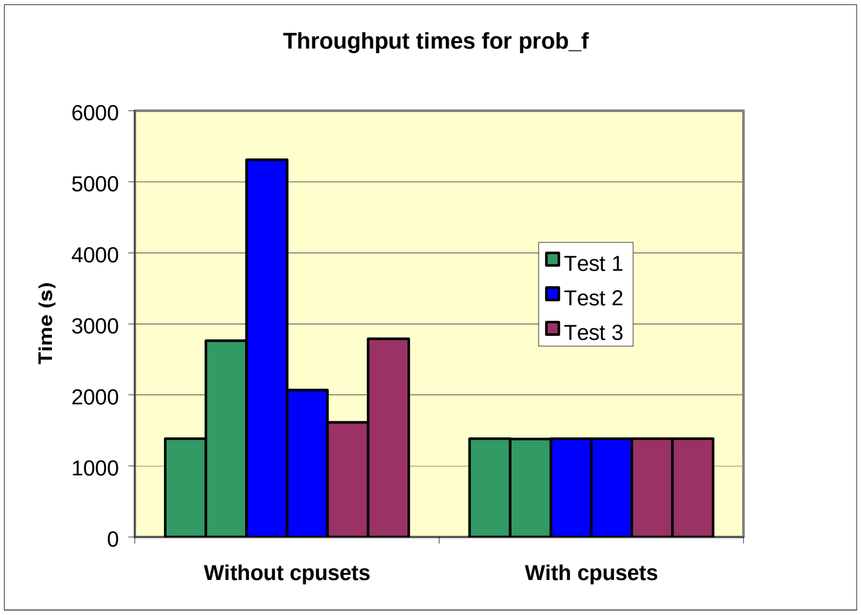

Figure 3. Throughput times for prob_f showing execution times for the two identical jobs that were part of each of the three throughput tests.

CODE 1				
# of processors	**with cpusets**	**without cpusets**		
		1	2	3
248	616	627	627	630
256	n/a	621	630	625

CODE 2				
# of processors	**with cpusets**	**without cpusets**		
		1	2	3
248	1676	1682	1684	1704
256	n/a	1730	1719	1725

Table 2. Times (in seconds) of tests of two codes run at large processor counts.